OpenXMプロジェクトの現状について (数式処理における理論と応用の研究)

AUTHOR(S):

奥谷, 行央; 小原, 功任; 高山, 信毅; 田村, 恭士; 野呂, 正行; 前川, 将秀

CITATION:

奥谷, 行央 ...[et al]. OpenXMプロジェクトの現状について (数式処理における理論と応用の研究). 数理解析研究所講究録 2000, 1138: 189-200

ISSUE DATE:

2000-04

URL:

http://hdl.handle.net/2433/63816

数理解析研究所講究録
1138巻 2000年 189-200

OpenXM プロジェクトの現状について

神戸大学大学院自然科学研究科 奥谷 行央(Yukio Okutani) *
金沢大学理学部 小原 功任(Katsuyoshi Ohara) †
神戸大学理学部 高山 信毅(Nobuki Takayama) ‡
神戸大学大学院自然科学研究科 田村 恭士(Takashi Tamura) §
富士通研究所 野呂 正行(Masayuki Noro) ¶
神戸大学理学部 前川 将秀(Masahide Maekawa) ‖

1 OpenXMとは

OpenXM は数学プロセス間でメッセージを交換するための規約である. 数学プロセス間でメッセージをやりとりすることにより, ある数学プロセスから他の数学プロセスを呼び出して計算を行なったり, 他のマシンで計算を行なわせたりすることが目的である. なお, OpenXM とは Open message eXchange protocol for Mathematics の略である. OpenXM の開発の発端は野呂と高山により, asir と kan/sm1 を相互に呼び出す機能を実装したことである.

初期の実装では, 相手側のローカル言語の文法に従った文字列を送っていた. この方法では相手側のソフトが asir なのか kan/sm1 なのかを判別するなどして, 相手側のローカル言語の文法に合わせた文字列を作成しなければならない. このローカル言語の文法に従った文字列を送る方法は, 効率的であるとはいい難いが, 使いやすいとも言える.

現在の OpenXM 規約では共通表現形式によるメッセージを用いている. 上記の文字列を送る方法の利点を生かすため, OpenXM 規約では共通表現形式の中の文字列として, ローカル言語の文法に従った文字列を用いたメッセージの交換も可能となっている.

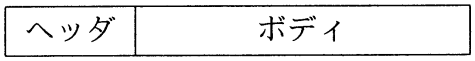
*okutani@math.sci.kobe-u.ac.jp
†ohara@kappa.s.kanazawa-u.ac.jp
‡takayama@math.sci.kobe-u.ac.jp
§tamura@math.sci.kobe-u.ac.jp
¶noro@para.flab.fujitsu.co.jp
‖maekawa@math.sci.kobe-u.ac.jp

OpenXM 規約では通信の方法に自由度があるが, 現在のところは TCP/IP を用いた通信しか実装されていない. [1] そこで, この論文では TCP/IP を用いた実装に準拠して OpenXM の説明をする.

2 OpenXM のメッセージの構造

通信の方法によってメッセージの構造は変わる. この論文では TCP/IP の場合についてのみ説明を行なう.

OpenXM 規約で規定されているメッセージはバイトストリームとなっており, 次のような構造になっている.



ヘッダの長さは 8 バイトであると定められている. ボディの長さはメッセージごとに異なっているが, 長さは 0 でもよい.

ヘッダは次の二つの情報を持っている.

1. 前半の 4 バイト. メッセージの種類を表す識別子であり, タグと呼ばれる.
2. 後半の 4 バイト. メッセージにつけられた通し番号である.

それぞれの 4 バイトは 32 ビット整数とみなされて扱われる.

この場合に用いられる 32 ビット整数の表現方法について説明しておこう. 問題になるのは負数の表現とバイトオーダーの問題である. まず, 負数を表す必要があるときには 2 の補数表現を使うことになっている. 次にバイトオーダーであるが, OpenXM 規約は複数のバイトオーダーを許容する. ただし一つの通信路ではひとつのバイトオーダーのみが許され, 通信路の確立時に一度だけ選ばれる.

現在の OpenXM 規約では, タグ (整数値) として以下のものが定義されている.

```
#define OX_COMMAND                  513
#define OX_DATA                     514
#define OX_SYNC_BALL                515
#define OX_DATA_WITH_LENGTH         521
#define OX_DATA_OPENMATH_XML        523
#define OX_DATA_OPENMATH_BINARY     524
#define OX_DATA_MP                  525
```

[1] ただし asir には MPI を用いた実装もある.

ボディの構造はメッセージの種類によって異なる. OX_COMMAND で識別されるメッセージはスタックマシンへの命令であり, それ以外のメッセージは何らかのオブジェクトを表している. この論文では OX_DATA と OX_COMMAND で識別されるメッセージについてのみ, 説明する.

既存のメッセージでは対応できない場合は, 新しい識別子を定義することで新しい種類のメッセージを作成することができる. この方法は各数学ソフトウェアの固有の表現を含むメッセージを作成したい場合などに有効である. 新しい識別子の定義方法については, [4] を参照すること.

3 OpenXM の計算モデル

OpenXM 規約での計算とはメッセージを交換することである. また, OpenXM 規約ではクライアント・サーバモデルを採用しているので, メッセージの交換はサーバとクライアントの間で行なわれる. [2] クライアントからサーバへメッセージを送り, クライアントがサーバからメッセージを受け取ることによって計算の結果が得られる. このメッセージのやりとりはクライアントの主導で行われる. つまり, クライアントは自由にメッセージをサーバに送付してもよいが, サーバからは自発的にメッセージが送付されることはない. この原理はサーバはスタックマシンであることで実現される. スタックマシンの構造については 4 節で述べる.

サーバがクライアントから受け取ったオブジェクト (つまり OX_COMMAND でないメッセージのボディ) はすべてスタックに積まれる. スタックマシンへの命令 (OX_COMMAND で識別されるメッセージのボディ) を受け取ったサーバは命令に対応する動作を行なう. このとき, 命令によってはスタックからオブジェクトを取り出すことがあり, また (各数学システムでの) 計算結果をスタックに積むことがある. もし, 与えられたデータが正しくないなどの理由でエラーが生じた場合にはサーバはエラーオブジェクトをスタックに積む. 計算結果をクライアントが得る場合にはスタックマシンの命令 SM_popCMO または SM_popString をサーバに送らなければならない. これらの命令を受け取ってはじめて, サーバからクライアントへメッセージが送られる.

まとめると, クライアントがサーバへメッセージを送り, 計算の結果を得るという手順は以下のようになる.

1. まず, クライアントがサーバへオブジェクトを送る. サーバは送られてきたオブジェクトをスタックに積む.

---

[2] 現在, 主に野呂が OpenXM の計算モデルの拡張を考えている. 効率的な分散計算のアルゴリズムの多くはサーバ同士の通信も要求するからである.

2. クライアントがサーバに計算の命令を送ると, サーバはあらかじめ定めれらた動作を行う. 一部の命令はスタックの状態を変更する. 例えば SM_executeFunction, SM_executeStringByLocalParser などの命令は, スタック上のオブジェクトから計算を行う. SM_popCMO もしくは SM_popString は, スタックの最上位のオブジェクトを取りだし, クライアントに送り返す.

4 OpenXM スタックマシン

OpenXM 規約ではサーバはスタックマシンであると定義している. 以下, OpenXM スタックマシンと呼ぶ. この節では OpenXM スタックマシンの構造について説明しよう.

まず, OpenXM 規約は通信時にやりとりされる共通のデータ形式については規定するが, OpenXM スタックマシンがスタックに積む, オブジェクトの構造までは規定しない. つまり, オブジェクトの構造は各数学システムごとに異なっているということである. このことは通信路からデータを受け取った際に, 各数学システムが固有のデータ構造に変換してからスタックに積むことを意味する. この変換は 1 対 1 対応である必要はない. もちろん, 恣意的に変換してよいわけではなく, 数学システムごとに変換方法をあらかじめ定めておく必要がある. このような共通のデータ形式と各システムでの固有のデータ形式との変換の問題は OpenXM に限ったことではない. OpenMath (8 節を参照のこと) ではこの変換を行うソフトウェアを Phrasebook と呼んでいる.

次に OpenXM スタックマシンの命令コードについて説明する. OpenXM スタックマシンにおけるすべての命令は 4 バイトの長さを持つ. OpenXM 規約の他の規定と同様に, 4 バイトのデータは 32 ビット整数と見なされるので, この論文でもその表記にしたがう. OpenXM スタックマシンに対する命令はスタックに積まれることはない. 現在のところ, OpenXM 規約では以下の命令が定義されている.

```
#define SM_popSerializedLocalObject          258
#define SM_popCMO                            262
#define SM_popString                         263
#define SM_mathcap                           264
#define SM_pops                              265
#define SM_setName                           266
#define SM_evalName                          267
#define SM_executeStringByLocalParser        268
#define SM_executeFunction                   269
#define SM_beginBlock                        270
#define SM_endBlock                          271
```

```
#define SM_shutdown                                    272
#define SM_setMathCap                                  273
#define SM_executeStringByLocalParserInBatchMode       274
#define SM_getsp                                       275
#define SM_dupErrors                                   276
#define SM_DUMMY_sendcmo                               280
#define SM_sync_ball                                   281
#define SM_control_kill                               1024
#define SM_control_to_debug_mode                      1025
#define SM_control_exit_debug_mode                    1026
#define SM_control_ping                               1027
#define SM_control_start_watch_thread                 1028
#define SM_control_stop_watch_thread                  1029
#define SM_control_reset_connection                   1030
```

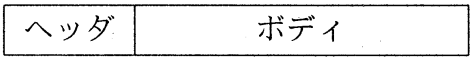
スタックマシンに対する命令の中には実行によって結果が返ってくるものがある. 結果が返ってくる命令を実行した場合, サーバはその結果をスタックに積む. たとえば, 命令 SM_executeStringByLocalParser はスタックに積まれているオブジェクトをサーバ側のローカル言語の文法に従った文字列とみなして計算を行なうが, 行なった計算の結果はスタックに積まれる.

なお, 命令の実行中にエラーが起こり, 結果が得られなかった場合には, エラーオブジェクトがスタックに積まれる.

5 CMO のデータ構造

OpenXM 規約では, 数学的オブジェクトを表現する方法として CMO 形式 (Common Mathematical Object format) を定義している. この CMO 形式にしたがったデータは, 識別子が OX_DATA であるようなメッセージのボディになることを想定している.

CMO 形式におけるデータ構造は次のような構造をもつ.



ヘッダは 4 バイトである. ボディの長さはそれぞれのデータによって異なるが, 0 でもよい.

メッセージと同様にヘッダは 4 バイト単位に管理される. すなわち, CMO ではヘッダは一つだけの情報を含む. この 4 バイトのヘッダのことをタグともいう. さて, CMO では, タグによってボディの論理的構造が決定する. すなわち, タグはそれぞれのデータ構造と

1 対 1 に対応する識別子である．それぞれの論理的構造は [4] に詳述されている．現在の OpenXM 規約では以下の CMO が定義されている．

```
#define CMO_ERROR2                          0x7f000002
#define CMO_NULL                            1
#define CMO_INT32                           2
#define CMO_DATUM                           3
#define CMO_STRING                          4
#define CMO_MATHCAP                         5
#define CMO_ARRAY                           16
#define CMO_LIST                            17
#define CMO_ATOM                            18
#define CMO_MONOMIAL32                      19
#define CMO_ZZ                              20
#define CMO_QQ                              21
#define CMO_ZERO                            22
#define CMO_DMS_GENERIC                     24
#define CMO_DMS_OF_N_VARIABLES              25
#define CMO_RING_BY_NAME                    26
#define CMO_RECURSIVE_POLYNOMIAL            27
#define CMO_LIST_R                          28
#define CMO_INT32COEFF                      30
#define CMO_DISTRIBUTED_POLYNOMIAL          31
#define CMO_POLYNOMIAL_IN_ONE_VARIABLE      33
#define CMO_RATIONAL                        34
#define CMO_64BIT_MACHINE_DOUBLE            40
#define CMO_ARRAY_OF_64BIT_MACHINE_DOUBLE   41
#define CMO_128BIT_MACHINE_DOUBLE           42
#define CMO_ARRAY_OF_128BIT_MACHINE_DOUBLE  43
#define CMO_BIGFLOAT                        50
#define CMO_IEEE_DOUBLE_FLOAT               51
#define CMO_INDETERMINATE                   60
#define CMO_TREE                            61
#define CMO_LAMBDA                          62
```

この中で CMO_ERROR2, CMO_NULL, CMO_INT32, CMO_DATUM, CMO_STRING, CMO_MATHCAP, CMO_LIST で識別されるオブジェクトは最も基本的なオブジェクトで

あって, すべての OpenXM 対応システムに実装されていなければならない.

これらについての解説を行う前に記法について, 少し説明しておく. この論文では, 大文字で CMO_INT32 と書いた場合には, 上記で定義した識別子を表す. また CMO_INT32 で識別されるオブジェクトのクラス (あるいはデータ構造) を cmo_int32 と小文字で表すことにする.

さて cmo を表現するための一つの記法を導入する. この記法は CMO expression と呼ばれている. その正確な形式的定義は [4] を参照すること.

CMO expssion は Lisp 風表現の一種で, cmo を括弧で囲んだリストとして表現する. それぞれの要素はカンマで区切る. 例えば,

(17, *int32*, (CMO_NULL), (2, *int32 n*))

は CMO expression である. ここで, 小文字の斜体で表された "*int32*" は 4 バイトの任意のデータを表す記号であり, "*int32 n*" は同じく 4 バイトのデータであるが以下の説明で n と表すことを示す. また数字 17, 2 などは 4 バイトのデータで整数値としてみたときの値を意味する. CMO_NULL は識別子 (すなわち数字 1 と等価) である. この記法から上記のデータは 20 バイトの大きさのデータであることが分かる. なお, CMO expression は単なる表記法であることに特に注意してほしい.

さて, この記法のもとで cmo_int32 を次のデータ構造であると定義する.

cmo_int32 := (CMO_INT32, *int32*)

同様に, cmo_null, cmo_string, cmo_list, cmo_mathcap のシンタックスは次のように定義される.

cmo_null := (CMO_NULL)
cmo_string := (CMO_STRING, *int32 n*, *string s*)
cmo_list := (CMO_LIST, *int32 m*, *cmo* c_1, ..., *cmo* c_m)
cmo_mathcap := (CMO_MATHCAP, *cmo_list*)

ただし, *string* は適当な長さのバイト列を表す. s のバイト長は n と一致することが要求される.

6 mathcap について

OpenXM 規約では, 通信時に用いられるメッセージの種類を各ソフトウェアが制限する方法を用意している. これは各ソフトウェアの実装によってはすべてのメッセージをサポートするのが困難な場合があるからである. また, 各ソフトウェアでメッセージの種類を拡張したい場合にも有効である. この制限 (あるいは拡張) は mathcap と呼ばれるデータ構造

によって行われる. この節では mathcap のデータ構造と, 具体的なメッセージの制限の手続きについて説明する.

まず, 手続きについて説明しよう.

第一にサーバの機能を制限するには次のようにする. クライアントが mathcap オブジェクトをサーバへ送ると, サーバは受け取った mathcap をスタックに積む. 次にクライアントが命令 SM_setMathCap を送ると, サーバはスタックの最上位に積まれている mathcap オブジェクトを取り出し, mathcap で設定されていないメッセージをクライアントへ送らないように制限を行う.

第二にクライアントを制限するには次のようにする. まず, クライアントがサーバに命令 SM_mathcap を送ると, サーバは mathcap オブジェクトをスタックに積む. さらに命令 SM_popCMO を送ると, サーバはスタックの最上位のオブジェクト (すなわち mathcap オブジェクト) をボディとするメッセージをクライアントに送付する. クライアントはそのオブジェクトを解析して, 制限をかける.

次に mathcap のデータ構造について説明する. mathcap は cmo の一種であるので, すでに説明したように

cmo_mathcap := (CMO_MATHCAP, *cmo_list*)

の構造をもつ (5 節を参照のこと). ボディは cmo_list オブジェクトでなければならない.

さて, mathcap オブジェクトのボディの cmo_list オブジェクトは以下の条件を満たすことを要求される. まず, その cmo_list オブジェクトは少なくともリスト長が 3 以上でなければならない.

(CMO_LIST, *int32*, *cmo a*, *cmo b*, *cmo c*, ...)

第一要素 a はまた cmo_list であり, リスト長は 4 以上, a_1 は cmo_int32 でバージョンを表す. a_2, a_3, a_4 は cmo_string であり, それぞれ数学システムの名前, バージョン, HOSTTYPE を表すことになっている.

(CMO_LIST, *int32*, *cmo_int32* a_1, *cmo_string* a_2, *cmo_string* a_3, *cmo_string* a_4, ...)

第二要素 b も cmo_list であり, OpenXM スタックマシンを制御するために用いられる. 各 b_i は cmo_int32 であり, ボディはスタックマシンの命令コードである. 4 節で説明したが, スタックマシンへの命令はすべて *int32* で表されていたことに注意しよう.

(CMO_LIST, *int32* n, *cmo_int32* b_1, ..., *cmo_int32* b_n)

第三要素 c は以下のような cmo_list であり, オブジェクトの送受信を制御するために用いられる. 送受信の制御はメッセージの種類ごとに行われる.

(CMO_LIST, *int32* m, *cmo_list* ℓ_1, ..., *cmo_list* ℓ_m)

各 ℓ_i が制御のための情報を表す. どの ℓ_i も一つ以上の要素を持っており, 第一要素は必ず cmo_int32 となっていなければならない. これは制御すべきメッセージの識別子を入れるためである.

各 ℓ_i の構造はメッセージの種類によって異なる. ここでは, OX_DATA の場合についてのみ説明する. 第一要素が OX_DATA の場合, リスト ℓ_i は以下のような構造となっている. 各 c_i は cmo_int32 であり, そのボディは CMO の識別子である. c_i で指示された CMO のみが送受信することを許される.

(CMO_LIST, 2, (CMO_INT32, OX_DATA),
(CMO_LIST, *int32* k, *cmo_int32* c_1, ..., *cmo_int32* c_k))

具体的な mathcap の例をあげよう. 名前が "ox_test", バージョンナンバーが 199911250 のサーバで, Linux 上で動いており, このサーバのスタックマシンが命令 SM_popCMO, SM_popString, SM_mathcap, SM_executeStringByLocalParser を利用可能で, かつ オブジェクトを cmo_int32, cmo_string, cmo_mathcap, cmo_list のみに制限したいときの mathcap は

(CMO_MATHCAP, (CMO_LIST, 3,
(CMO_LIST, 4, (CMO_INT32, 199911250), (CMO_STRING, 7, "ox_test"),
(CMO_STRING, 9, "199911250"), (CMO_STRING, 4, "i386"))
(CMO_LIST, 5, (CMO_INT32, SM_popCMO),
(CMO_INT32, SM_popString), (CMO_INT32, SM_mathcap),
(CMO_INT32, SM_executeStringByLocalParser))
(CMO_LIST, 1, (CMO_LIST, 2, (CMO_INT32, OX_DATA),
(CMO_LIST, 4, (CMO_INT32, CMO_INT32),
(CMO_INT32, CMO_STRING), (CMO_INT32, CMO_MATHCAP),
(CMO_INT32, CMO_LIST))))))

になる.

7 セキュリティ対策

OpenXM 規約は TCP/IP を用いて通信を行うことを考慮している. したがってネットワークによって接続される現代の多くのソフトウェアと同様, OpenXM 規約もまた通信時のセキュリティについて注意している. 以下, このことについて説明しよう.

第一に OpenXM では侵入者に攻撃の機会をできるだけ与えないようにするため, サーバは接続が必要になった時のみ起動している. しかし, これだけでは接続を行なう一瞬のすき

を狙われる可能性もある. そこで接続を行なう時に, 接続を行なうポート番号を毎回変えている. こうすることで, 特定のポート番号を狙って接続を行なう手口を防ぐことができる.

さらにもう一段安全性を高めるために, 接続時に一時パスワードをクライアントが作成し, そのパスワードを使って認証を行なう. このパスワードは一旦使用されれば無効になるので, もし仮になんらかの手段でパスワードが洩れたとしても安全である.

なお, メッセージ自体には特に暗号化などの処置を行っていないので, そのままではパケット盗聴などを受ける可能性がある. 現在の実装では, 必要ならば ssh を利用して対応している.

8 OpenXM 以外のプロジェクト

OpenXM 以外にも数式処理システム間の通信や数学データの共通表現を目指したプロジェクトは存在する. ここでは他のプロジェクトについても触れておこう.

- ESPRIT OpenMath Project

  http://www.nag.co.uk/projects/openmath/omsoc/

  数学的対象の SGML 的表記の標準化を目指した大規模なプロジェクト. このプロジェクトでは数学データを数学的意味を保ったままで如何に表現すべきかという問題を追求している. したがって既存の表現, 例えば TEX による数式の表現と OpenMath による数式の表現とでは, 本質的に意味が異なる. OpenMath で定義された表現は, 異なる種類の数式処理システムの間で情報を交換するときに利用することができる. しかしながら, 数学システム同士の通信, 例えばある数学システムから別の数学システムを呼び出して計算させる方法などは, このプロジェクトの対象外である. OpenXM における共通データ形式と数学システム固有のオブジェクトとの変換は OpenMath 規約の Phrasebook と同じアイデアを用いている.

- NetSolve

  http://www.cs.utk.edu/netsolve/

  NetSolve はクライアント・サーバ型の分散システムであり, 単なる計算システム以上のものを目指している. クライアントは必要に応じて, サーバを呼び出して計算をさせる. NetSolve の特徴は, サーバの呼び出しに Agent というソフトウェアを介在させることである. Agent は呼び出し先などを決定するデータベース的役割を果たす. また Agent によって負荷分散が可能になる. 現在の NetSolve は RPC を基礎にして実装されている.

- MP (Multi Project)

http://symbolicnet.mcs.kent.edu/SN/areas/protocols/mp.html

数学的なデータの共通表現を提供するプロジェクト. MP の主な関心は, この共通表現の最適化である. 数学システム間で, 命令を送信したりデータを受け渡す仕組み (control integration) は, このプロジェクトの対象外である. MP は既存の control integration に対して補完的役割を果たす.

MP では数式を構文木の一種 (annotated syntax tree) と捉える. annotated syntax tree には数学的な意味を保ったまま表現されているという特徴がある (この点は OpenMath と似ている). MP が提供する共通表現とは, この構文木のバイナリエンコーディング, つまりバイト列での表現のことである. MP の定義する表現ではバイト列の長さが最適化されている. また, バイトオーダーの選択も可能である (2 節参照のこと).

このプロジェクトでは C 言語および GNU Common Lisp で実装を行なっている. C 言語による実装 (MP-C ライブラリ) は上記のウェブページから収得可能である. このライブラリを用いて通信を行なうには, なんらかの control integration が必要である. control integration としては, ソケット, MPI, PVM などが利用できる.

- MCP (Mathematical Computation Protocol)

  http://horse.mcs.kent.edu/~pwang/

  数学的なデータや命令を含むメッセージをやりとりするための HTTP に似たプロトコル. MCP は control integration であり, クライアント・サーバ型の通信モデルを採用している. MCP のメッセージはヘッダとボディから構成されている. ヘッダはテキストであり, 最初に現れる空行でヘッダとボディは区切られている.

  数式はボディに記述されて送られる. 数式の表現方法としては MP や OpenMath で定められたものを使用することが考えられている. 実際, 数式の表現に OpenMath 規約の XML 表現を用いた実装があり, GAP と Axiom の間で通信が行なわれている. この場合, MCP によって送信されるメッセージはボディに OpenMath 形式で数式を記述したテキストである.

9 現在提供されているソフトウェア

現在 OpenXM 規約に対応しているクライアントには asir, sm1, Mathematica がある. これらのクライアントから OpenXM 規約に対応したサーバを呼び出すことができる. また OpenXM 規約に対応しているサーバには, asir, sm1, Mathematica, gnuplot, PHC pack などがあり, それぞれ ox_asir, ox_sm1, ox_math, ox_sm1_gnuplot, ox_sm1_phc という名

前で提供されている. さらに OpenMath 規約の XML 表現で表現されたオブジェクトと CMO 形式のオブジェクトを相互変換するソフトウェアが JAVA によって実装されており, OMproxy という名前で提供されている.

参 考 文 献

[1] O. Caprotti, A. M. Cohen: The OpenMath Standard, February 1999. (http://www.nag.co.uk/projects/OpenMath/omstd/partI.ps.gz)

[2] H. Casanova, J. Dongarra, A. Karainov, J. Wasniewski: Users' Guide to NetSolve, October 27, 1998. (http://www.cs.utk.edu/netsolve/download/ug.ps)

[3] S. Gray, N. Kajler, P. S. Wang: Design and Implementation of MP, a Protocol for Efficient Exchange of Mathematical Expressions, *Journal of Symbolic Computation*, **25**, February 1998, 213–238. (ftp://ftp.mcs.kent.edu/dist/MP/mp-jsc-96.ps.gz)

[4] 野呂 正行, 高山 信毅: Open XM の設計と実装 — Open message eXchange protocol for Mathematics, December 31, 1999. (http://www.math.sci.kobe-u.ac.jp/openxm/openxm-jp.tex)

[5] 小原 功任, 高山 信毅, 野呂 正行: Open asir 入門, 数式処理, **Vol 7**(No 2), 1999, 2–17. (ISBN 4-87243-086-7, SEG 出版, Tokyo).

[6] P. S. Wang: Design and Protocol for Internet Accessible Mathematical Computation, *Proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation*, 1999, 291–298. (ISBN 1-58113-073-2, ACM, New York 1999.).